Регулярные выражения в javascript

Оригинал: <http://www.pcre.ru/docs/javascript/text/intro1/>

<http://www.pcre.ru/docs/javascript/text/intro2/>

Общее описание

Регулярные выражения представляют собой образцы для поиска заданных

комбинаций символов в текстовых строках (такой поиск называется

сопоставлением с образцом). Существует два способа присваивания

переменным регулярных выражений, а именно:

\* Использование инициализатора объекта: var re = /pattern/switch?.

\* Использование конструктора RegExp: var re = new

RegExp("pattern"[,"switch"]?).

Здесь pattern - регулярное выражение, а switch - необязательные

опции поиска.

Инициализаторы объекта, например, var re = /ab+c/, следует применять в

тех случаях, когда значение регулярного выражения остается неизменным

во время работы сценария. Такие регулярные выражения компилируются в

процессе загрузки сценария и, следовательно, выполняются быстрее.

Вызов конструктора, например, var re = new RegExp("ab+c"), следует

применять в тех случаях, когда значение переменной будет меняться.

Если вы собираетесь использовать регулярное выражение несколько раз,

то имеет смысл скомпилировать его методом compile для более

эффективного поиска образцов.

При создании регулярного выражения следует учитывать, что заключение

его в кавычки влечет за собой необходимость использовать

escape-последовательности, как и в любой другой строковой константе.

Например, следующие два выражения эквивалентны:

var re = /\w+/g;

var re = new RegExp("\\w+", "g"); // В строке "\" должно заменяться на "\\"

Примечание: регулярное выражение не может быть пустым: два символа //

подряд задают начало комментария. Поэтому для задания пустого

регулярного выражения используйте выражение /.?/.

Регулярные выражения используются методами exec и test объекта RegExp

и методами match, replace, search и split объекта String. Если нам

нужно просто проверить, содержит ли данная строка подстроку,

соответствующую образцу, то используются методы a

href="/cgi-bin/print.pl?id=118-4.html#505" class=intext>test или

search. Если же нам необходимо извлечь подстроку (или подстроки),

соответствующие образцу, то нам придется воспользоваться методами exec

или match. Метод replace обеспечивает поиск заданной подстроки и

замены ее на другую строку, а метод split позволяет разбить строку на

несколько подстрок, основываясь на регулярном выражении или обычной

текстовой строке. Более подробные сведения о применении регулярных

выражений приведены в описании соответствующих методов.

**Синтаксис регулярных выражений**

Регулярное выражение может состоять из обычных символов; в этом случае

оно будет соответствовать заданной комбинации символов в строке.

Например, выражение /ком/ соответствует выделенным подстрокам в

следующих строках: "комок", "лакомка", "главком флота". Однако,

гибкость и мощь регулярным выражениям придает возможность

использования в них специальных символов, которые перечислены в

следующей таблице.

Специальные символы в регулярных выражениях:

\

Для символов, которые обычно трактуются буквально, означает, что

следующий символ является специальным. Например, /n/ соответствует

букве n, а /\n/ соответствует символу перевода строки.

Для символов, которые обычно трактуются как специальные, означает, что

символ должен пониматься буквально. Например, /^/ означает начало

строки, а /\^/ соответствует просто символу ^. /\\/ соответствует

обратной косой черте \.

^

Соответствует началу строки.

$

Соответствует концу строки.

\*

Соответствует повторению предыдущего символа нуль или более раз.

+

Соответствует повторению предыдущего символа один или более раз.

?

Соответствует повторению предыдущего символа нуль или один раз.

.

Соответствует любому символу, кроме символа новой строки.

(pattern)

Соответствует строке pattern и запоминает найденное соответствие.

(?:pattern)

Соответствует строке pattern, но не запоминает найденное соответствие.

Используется для группировки частей образца, например, /ко(?:т|шка)/ -

это краткая запись выражения /кот|кошка/.

(?=pattern)

Соответствие с "заглядыванием вперед", происходит при соответствии

строки pattern без запоминания найденного соответствия. Например,

/Windows (?=95|98|NT|2000)/ соответствует "Windows " в строке "Windows

98", но не соответствует в строке "Windows 3.1". После сопоставления

поиск продолжается с позиции, следующей за найденным соответствием,

без учета заглядывания вперед.

(?!pattern)

Соответствие с "заглядыванием вперед", происходит при несоответствии

строки pattern без запоминания найденного соответствия. Например,

/Windows (?!95|98|NT|2000)/ соответствует "Windows " в строке "Windows

3.1", но не соответствует в строке "Windows 98". После сопоставления

поиск продолжается с позиции, следующей за найденным соответствием,

без учета заглядывания вперед.

x|y

Соответствует x или y.

{n}

n - неотрицательное число. Соответствует ровно n вхождениям

предыдущего символа.

{n,}

n - неотрицательное число. Соответствует n или более вхождениям

предыдущего символа. /x{1,}/ эквивалентно /x+/. /x{0,}/ эквивалентно

/x\*/.

{n,m}

n и m - неотрицательное числа. Соответствует не менее чем n и не более

чем m вхождениям предыдущего символа. /x{0,1}/ эквивалентно /x?/.

[xyz]

Соответствует любому символу из заключенных в квадратные скобки.

[^xyz]

Соответствует любому символу, кроме заключенных в квадратные скобки.

[a-z]

Соответствует любому символу в указанном диапазоне.

[^a-z]

Соответствует любому символу, кроме лежащих в указанном диапазоне.

\b

Соответствует границе слова, т. е. позиции между словом и пробелом или

переводом строки.

\B

Соответствует любой позиции, кроме границе слова.

\сX

Соответствует символу Ctrl+X. Например, /\cI/ эквивалентно /\t/.

\d

Соответствует цифре. Эквивалентно [0-9].

\D

Соответствует нецифровому символу. Эквивалентно [^0-9].

\f

Соответствует символу перевода формата (FF).

\n

Соответствует символу перевода строки (LF).

\r

Соответствует символу возврата каретки (CR).

\s

Соответствует символу пробела. Эквивалентно /[ \f\n\r\t\v]/.

\S

Соответствует любому непробельному символу. Эквивалентно /[^ \f\n\r\t\v]/.

\t

Соответствует символу табуляции (HT).

\v

Соответствует символу вертикальной табуляции (VT).

\w

Соответствует латинской букве, цифре или подчеркиванию. Эквивалентно /[A-Za-z0-9\_] /.

\W

Соответствует любому символу, кроме латинской буквы, цифры или

подчеркивания. Эквивалентно /[^A-Za-z0-9\_] /.

\n

n - положительное число. Соответствует n-ной запомненной подстроке.

Вычисляется путем подсчета левых круглых скобок. Если левых скобок до

этого символа меньше, чем n, то эквивалентно \0n.

\0n

n - восьмеричное число, не большее 377. Соответствует символу с

восьмеричным кодом n. Например, /\011/ эквивалентно /\t/.

\xn

n - шестнадцатеричное число, состоящее из двух цифр. Соответствует

символу с шестнадцатеричным кодом n. Например, /\x31/ эквивалентно /1/.

\un

n - шестнадцатеричное число, состоящее из четырех цифр. Соответствует

символу Unicode с шестнадцатеричным кодом n. Например, /\u00A9/

эквивалентно /©/.

Регулярные выражения вычисляются аналогично остальным выражениям

JavaScript, т. е. с учетом приоритета операций: операции, имеющие

больший приоритет, выполняются первыми. Если операции имеют равный

приоритет, то они выполняются слева направо. В следующей таблице

приведен список операций регулярных выражений в порядке убывания их

приоритетов; операции, расположенные в одной строке таблицы, имеют

равный приоритет.

Операции:

\

() (?:) (?=) (?!) []

\* + ? . {n} {n,} {n,m}

^ $ \метасимвол

|

**Опции поиска**

При создании регулярного выражения мы можем указать дополнительных

опции поиска:

\* i (ignore case). Не различать строчные и прописные буквы.

\* g (global search). Глобальный поиск всех вхождений образца.

\* m (multiline). Многострочный поиск.

\* Любые комбинации этих трех опций, например ig или gim.

Приведем несколько примеров. Поскольку регулярные выражения различают

строчные и прописные буквы, следующий сценарий

var s = "Изучаем язык JavaScript";

var re = /JAVA/;

var result = re.test(s) ? "" " : "" не ";

document.write("Строка "" + s + result + "соответствует образцу " + re);

выведет в окно обозревателя текст:

Строка "Изучаем язык JavaScript" не соответствует образцу /JAVA/

Если мы теперь заменим вторую строку примера на var re = /JAVA/i;, то

на экран будет выведен текст:

Строка "Изучаем язык JavaScript" соответствует образцу /JAVA/i

Теперь рассмотрим опцию глобального поиска. Она обычно применяется

методом replace при поиске образца и замены найденной подстроки на

новую. Дело в том, что по умолчанию этот метод производит замену

только первой найденной подстроки и возвращает полученный результат.

Рассмотрим следующий сценарий:

var s = "Мы пишем сценарии на JavaScript, " +

"но JavaScript - не единственный сценарный язык.";

var re = /JavaScript/;

document.write(s.replace(re, "VBScript"));

Он выводит в окно обозревателя текст, который явно не соответствует

желаемому результату:

Мы пишем сценарии на VBScript, но JavaScript - не единственный

сценарный язык.

Для того, чтобы все вхождения строки "JavaScript" были заменены на

"VBScript", мы должны изменить значение регулярного выражения на var

re = /JavaScript/g;. Тогда результирующая строка будет иметь вид:

Мы пишем сценарии на VBScript, но VBScript - не единственный сценарный

язык.

Наконец, опция многострочного поиска позволяет проводить сопоставление

с образцом строкового выражения, состоящего из нескольких строк

текста, соединенных символами разрыва строки. По умолчанию,

сопоставление с образцом прекращается, если найден символ разрыва

строки. Данная опция преодолевает указанное ограничение и обеспечивает

поиск образца по всей исходной строке. Она также влияет на

интерпретацию некоторых специальных символов в регулярных выражениях,

а именно:

\* Обычно символ ^ сопоставляется только с первым элементом строки.

Если же опция многострочного поиска включена, то он также

сопоставляется с любым элементом строки, которому предшествует

символ разрыва строки.

\* Обычно символ $ сопоставляется только с последним элементом

строки. Если же опция многострочного поиска включена, то он также

сопоставляется с любым элементом строки, который является символом

разрыва строки.

**Запоминание найденных подстрок**

Если часть регулярного выражения заключена в круглые скобки, то

соответствующая ей подстрока будет запомнена для последующего

использования. Для доступа к запомненным подстрокам используются

свойства $1, :, $9 объекта RegExp или элементы массива, возвращаемого

методами exec и match. В последнем случае количество найденных и

запомненных подстрок не ограничено.

Например, следующий сценарий использует метод replace для перестановки

слов в строке. Для замены найденного текста используются свойства $1 и $2.

var re = /(\w+)\s(\w+)/;

var str = "Михаил Булгаков";

document.write(str.replace(re, "$2, $1"))

Этот сценарий выведет в окно обозревателя текст:

Булгаков, Михаил

[http://program.rin.ru](http://program.rin.ru/),

15.05.2006.

Часть 2

Введение

Регулярные выражения -- это мощное средство для обработки входящих

данных. Задача, требующая замены или поиска текста, может быть красиво

решена с помощью этого "языка внутри языка". И хотя максимальный

эффект от регулярных выражений можно добиться при использования

серверных языков, всё же не стоит недооценивать возможности этого

приложения и на стороне клиента.

Основные понятия

Регулярное выражение (regular expression) -- средство для обработки

строк или последовательность символов, определяющая шаблон текста.

Модификатор -- предназначен для "инструктирования" регулярного

выражения.

Метасимволы -- специальные символы, которые служат командами языка

регулярных выражений.

Регулярное выражение задаётся как обычная переменная, только вместо

кавычек используется слеш, например: var reg=/рег\_выражение/

Под простейшими шаблонами будем понимать такие шаблоны, которые не

нуждаются в каких-либо специальных символах.

Допустим, нашей задачей является замена всех букв "р" (малых и

заглавных) на латинскую большую букву "R" в словосочетании Регулярные

выражения.

Создаём шаблон var reg=/р/ и воспользуясь методом replace осуществляем

задуманное

<script language="JavaScript">

var str="Регулярные выражения"

var reg=/р/

var result=str.replace(reg, "R")

document.write(result)

</script>

В результате получим строку <<РегуляRные выражения>>, замена произошла

только на первом вхождении буквы "р" с учётом регистра.

Но под условия нашей задачи этот результат не подходит... Тут нам

понадобятся модификаторы "g" и "i", которые могут использоваться как

отдельно, так и совместно. Эти модификаторы ставятся в конце шаблона

регулярного выражения, после слэша, и имеют следующие значения:

модификатор "g" -- задаёт поиск в строке как "глобальный", т.е. в

нашем случае замена произойдет для всех вхождений буквы "р". Теперь

шаблон выглядит так: var reg=/р/g, подставив его в наш код

<script language="JavaScript">

var str="Регулярные выражения"

var reg=/р/g

var result=str.replace(reg, "R")

document.write(result)

</script>

получим строку <<РегуляRные выRажения>>.

модификатор "i" -- задаёт поиск в строке без учёта регистра, добавив

этот модификатор в наш шаблон var reg=/р/gi, после выполнения скрипта

получим искомый результат нашей задачи -- <<RегуляRные выRажения>>.

**Специальные символы (метасимволы)**

Метасимволы задают тип символов искомой строки, способ окружения

искомой строки в тексте, а так же количество символов отдельного типа

в просматриваемом тексте. Поэтому метасимволы можно разделить на три

группы:

\* Метасимволы поиска совпадений.

\* Количественные метасимволы.

\* Метасимволы позиционирования.

Метасимволы поиска совпадений

\b

граница слова, задаёт условие, при котором шаблон должен выполняться в

начале или конце слов.

\B

не граница слова, задаёт условие, при котором шаблон не выполняется в

начале или конце слова.

\d, цифра от 0 до 9.

\D, не цифра.

\s, одиночный пустой символ, соответствует символу пробела.

\S, одиночный непустой символ, любой один символ за исключением

пробела.

\w, буква, цифра или символ подчёркивания.

\W, не буква, цифра или символ подчёркивания.

.

любой символ, любые знаки, буквы, цифры и т.д.

[ ]

набор символов, задаёт условие, при котором шаблон должен выполняться

при любом совпадении символов заключенных в квадратные скобки.

[^ ], набор не входящих символов, задаёт условие, при котором шаблон

не должен выполняться при любом совпадении символов заключенных в

квадратные скобки.

Количественные метасимволы

\*

ноль и большее количество раз.

?

Ноль или один раз

+

Один и большее количество раз.

{n}

точно n раз.

{n,}

n или большее количество раз.

{n,m}

по крайней мере, n раз, но не более чем m раз.

Метасимволы позиционирования

^

в начале строки.

$

в конце строки.

**Некоторые методы для работы с шаблонами**

replace -- данный метод мы уже использовали в самом начале статьи, он

предназначен для поиска образца и замены найденной подстроки на новую

подстроку.

test -- данный метод проверяет, есть ли совпадения в строке

относительно шаблона и возвращает false, если сопоставление с образцом

закончилось неудачей, в противном случае true.

например:

<script language="JavaScript">

var str="JavaScript"

var reg=/PHP/

var result=reg.test(str)

document.write(result)

</script>

выведет в качестве результата false, т.к. строка "JavaScript" не содержит

строку "PHP".

Затем значение возвращаемое методом test может быть преобразовано

программистом из true или false в любую другую строку

например:

<script language="JavaScript">

var str="JavaScript"

var reg=/PHP/

var result=reg.test(str) ? "Строка совпала" : "Строка не совпала"

document.write(result)

</script>

в этом случае в качестве результата будет строка "Строка не

совпала".

exec -- данный метод выполняет сопоставление строки с образцом,

заданным шаблоном. Если сопоставление с образцом закончилось неудачей,

то возвращается значение null. В противном случае результатом является

массив подстрок, соответствующих заданному образцу. /\*Первый элемент

массива будет равен исходной строке удовлетворяющее заданному

шаблону\*/

например:

<script language="JavaScript">

var reg=/(\d+).(\d+).(\d+)/

var arr=reg.exec("Я родился 15.09.1980")

document.write("Дата рождения: ", arr[[ ,]] "< br>")

document.write("День рождения: ", arr[,](http://www.pcre.ru/rss/examples/) "< br>")

document.write("Месяц рождения: ", arr[,](http://www.pcre.ru/rss/docs/) "< br>")

document.write("Год рождения: ", arr[,](http://www.pcre.ru/) "< br>")

</script>

в результате получим четыре строки:

Дата рождения: 15.09.1980

День рождения: 15

Месяц рождения: 09

Год рождения: 1980

**Заключение**

В статье отображено далеко не все возможности и прелести регулярных

выражений, для более глубокого изучения этого вопроса посоветую

изучить объект RegExp. Так же хочу обратить внимание на то, что

синтаксис регулярных выражений не чем не отличается как в JavaScript,

так и в PHP. К примеру, для проверки правильности ввода e-mail,

регулярное выражение, что для JavaScript, что для PHP будет выглядеть

одинаково /[0-9a-z\_]+@[0-9a-z\_^.]+.[a-z]{2,3}/i.

Выскорко М. С.,

15.05.2006.